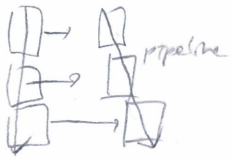
Prog. Model 1 : Sequential (SISD)

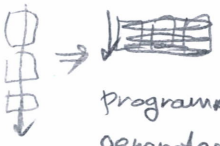


↳ pipelined processor

Out of order execution processor

(← the loop is dynamically unrolled) by the hardware

Prog. Model 2 : Data Parallel (SIMD)

↳ 

Programmer or Compiler generates SIMD instruction!

→ Best executed by SIMD processor

Prog. Model 3 : Multi-threaded

↳ 

programmer or compiler generates a thread to execute each iteration.

→ By MIMD machine.

→ AKA ; SPMD

(Single program multiple data)

A GPU is a SIMD (SIMT) machine

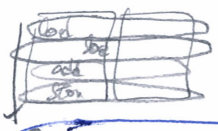
↳ not programmed using "SIMD instr"

- Programmed using threads (SPMD)
- A set of thread executing same instruction are dynamically grouped into a warp (wavefront) by the hardware.

→ Warp = SIMD operation by the hardware

(set of threads executing same instr)

- SIMT ; Single instruction multiple threads.



SPMD on SIMT machine

- SIMD : A single sequential INS stream of SIMD instr.

[VLD, VLD, VADD, VST], VLEN

- SIMT : multiple INS streams of scalar instr

→ grouped to warp

[LD, LD, ADD, ST], NumThread.

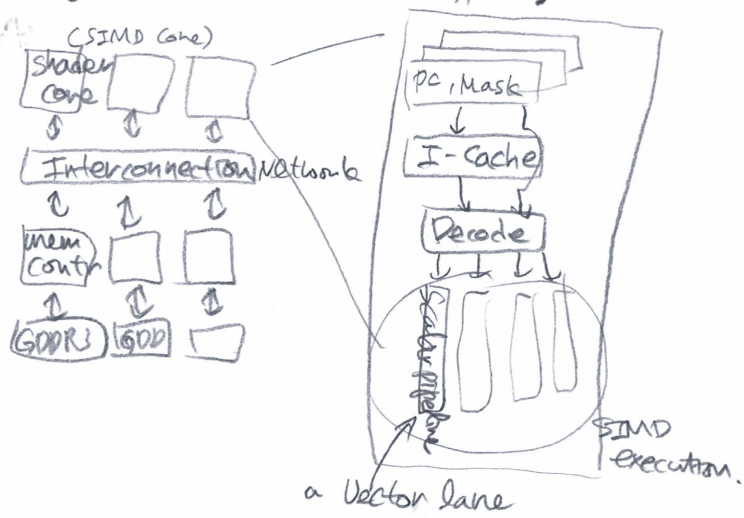
Assume a warp = 32 threads

if N = 32K → 1000 warps

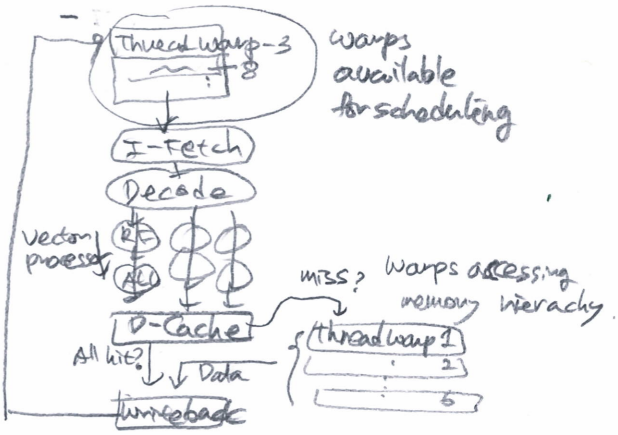
(1 iter/thread)

- Warps can be interleaved on the same pipeline → Fine grained multithreading of warps.

< High level view of GPU >



Latency hiding via Warp-level FGMT.



Lect.22 GPU Programming

- Nvidia GeForce GTX 285 (2009)
  - Nvidia-speak: 240 stream processors "SIMT" execution
  - Generic-speak: 30 cores, 8 SIMD functional units per core (=8 vector lanes)

32 thread → warp ⟶ 32 warps in FGMT manner.

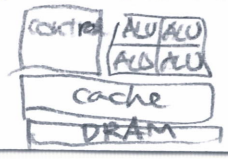
- Nvidia V100 → 5120 stream processor = 80 cores w/ 64 SIMD func. units
  - ⊕ Specialized Functional Units for machine learning ("tensor"-cores)

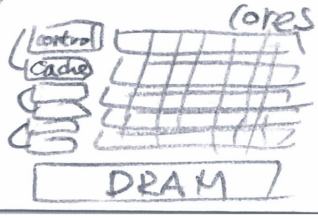
15.7 TFLOPS : single precision
7.8 TFLOPS : double prec.
125 ~~TFlops~~ for Deep learning ("Tensor" cores)

- Inherent Parallelism → Matrices, Deep learning, Image processing
  - New programming tool

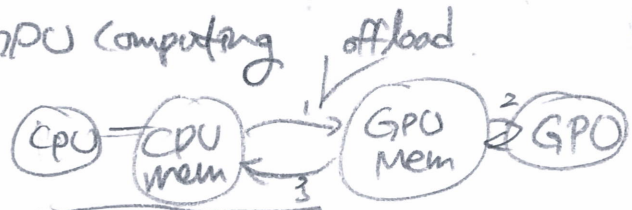
- Bottleneck
  - CPU-GPU data transfer (PCIe, NVLINK)
  - DRAM memory bandwidth (GDDR5, HBM2)
    - ⊕ Data layout.

- CPU vs GPU
  - few out-of-order cores
  - many in-order FGMT cores.





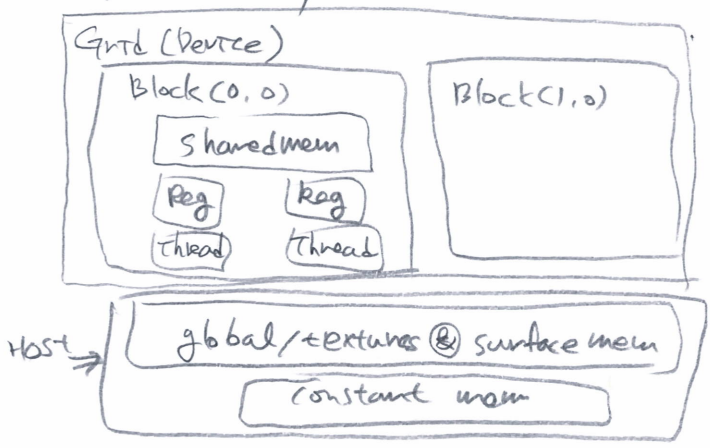
- GPU computing



serial code (host)
↓
parallel kernel (device)
↓
serial code (host)
↓
parallel kernel (device)
⋮

- **CUDA / OpenCL** programming model.
  - Bulk synchronous programming
    - ↳ Global (coarse-grain) sync.
  - Host (typically CPU) allocates memory, copies data, launch kernels.
  - Device (GPU) executes kernels
    - ↳ Grid (NDRange) → Block (work-group) → Thread (work-item)

- Memory Hierarchy



• CUDA programming

- Fkn prototypes : float serialFunction(...);
  __global__ void kernel(...);
- main()
  1) Allocate memory → cudaMalloc
  2) Data from Host to Device → cudaMemCpy
  3) Set #blocks & # threads
  4) Kernel Call ⇒ kernel<<<execution configuration>>>(args);
  5) Result from Device to Host → cudaMemCpy

repeat as needed.

- Kernel ⇒ __global__ void kernel( )
  ↳ registers
  Shared memory : __shared__
  Intra-block synch : __syncthreads()
- Mem deallocation ⇒ cudaFree(d_in);
- Explicit sync → cudaDeviceSynchronize();

• Image layout = (ex) Row-major layout in (1D)
  ↳ one GPU thread per pixel
  Grid of Blocks of Threads
  ⇒ gridDim.x , blockDim.x
  ⊕ blockIdx.x , threadIdx.x
  ↳ pixel address = blockIdx.x ⊗ blockDim.x + threadIdx.x

• (2D) Grid ⇒ gridDim.x , gridDim.y
  blockDim.x , blockDim.y
  blockInd.x , blockDim.y

[Memory Access]

① Latency Hiding w/ FGMT
② Occupancy
③ Memory Coalescing
  AoS vs SoA
  (Array of Structure) (Structure of Array)
④ Data reuse : Tiling → shared memory.
  ↳ __shared__ int l_data[L_SIZE+2)*(L_SIZE+2)];
⑤ Shared memory = banked (interleaved) memo
  ↳ 32 banks; Bank = Address % 32
  ↳ BankConflict → Padding
  Randomized mapping
  Hash fkn

[SIMD Utilization]

• Intra warp divergence
  ↳ (ex) Vector reduction : Naïve Mapping
  ↓
  Divergence free mapping

[• Atomic operations]

• Atomic conflicts
  Histogram Calculation..
• Privatization : Per-block sub-histograms in shared memory

[• Data Transfers] - Synchronous
  Asynchronous
  ↳ Divide into N streams.
  ↳ Video processing

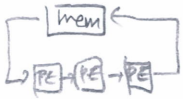
Lect23a. Systolic Arrays and Beyond

Goal: Design an accelerator that has
- Simple / Regular Design / High Concurrency / Balanced Computation & I/O (mem) bandwidth

Idea: Replace a single PE with a regular array of PEs and carefully orchestrate flow of data between PEs

Schematic:

instead of this

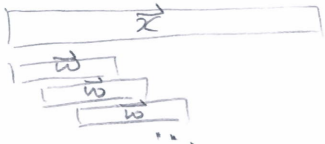


Systolic ⟨ MISD

(memory → heart, Data → blood, PEs → cells)

- Differences in pipelining → Array of PEs (Can be non-linear & multi-dimensional)

ex) Convolution.



$y_1 = w_1x_1 + w_2x_2 + w_3x_3 + \dots$

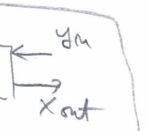
$y_2 = w_1x_2 + w_2x_3 + w_3x_4 + \dots$

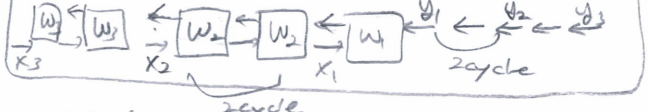
$\vdots$

⇒

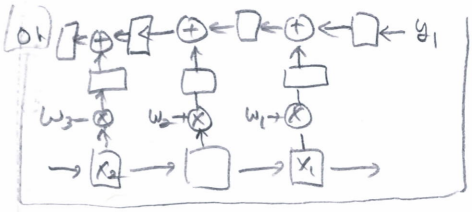
$y_{out} = y_{in} + w \cdot x_{in}$

$x_{out} = x_{in}$





w_i's stay & x_i & y_i moves systolically in opposite direction.



- Carefully Orchestrate ⚠
- Two Dimensional Systolic Arrays & Combination of two different Systolic Array.

Advantages → Principled: w/ limited memory bandwidth balances comp. to I/O

Specialize: efficiency, simple, high concurrency.

Downside → Specialize

more generality ⇒ multiple weights in a PE + Data memory in PE (temp. result)

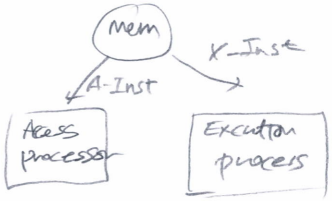
→ pipeline-parallel Program

→ put stages in a loop

→ a pipeline.

- The Warp Computer.
- Modern Systolic Array : TPU

Decoupled Access / Execute (DAE)



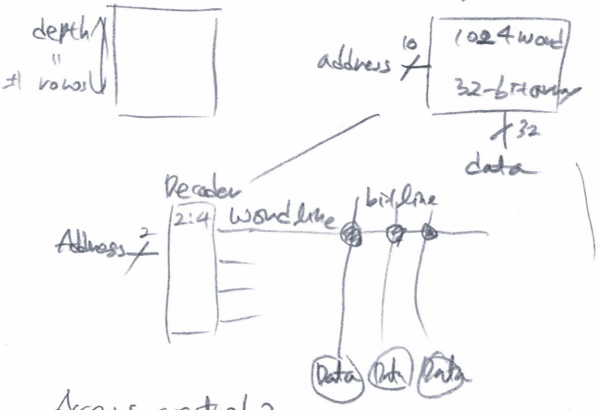
Lec 23 b. Memory Organization & Technology

- System maps virtual mem. addresses to physical mem.

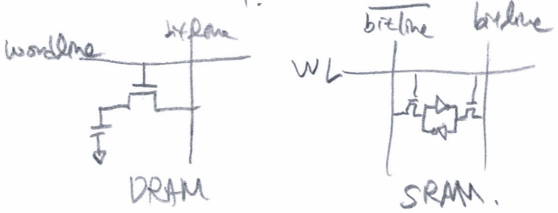
- Ideal Sys: Instr → Pipeline ← Data Supply

- **Devices**
  - Flip-Flops (or Latches); fast, expensive, parallel >10 Tr
  - Static-RAM; fast, expensive, one data word =6 Tr at a time
  - Dynamic-RAM; slower, one data at a time. refresh, cheap 1 Tr + 1 Cap
  - Storage (Flash, HDD); very cheap, non-volatile

- **Array** = ① a memory array + ② Address selection logic + ③ Readout circuitry



Access control?



- **Make larger?** → Divide the mem into smaller arrays.

DRAM ⇒ Channel → Rank → Bank → Subarrays → Mats ⇝ Interleaving

SRAM
1. address decode
2. drive row select → latency
3. selected bit-cells drive bitlines (entire row is read together)
4. Differential Sensing & Column Select
5. precharge all bitlines

DRAM

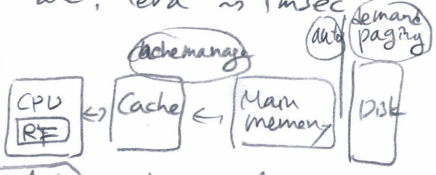
Read → ① address decode
- ② drive row select
- ③ selected bit-cell drive bitlines
- ④ A "flip-flopping" sense amplifier & regen the bitline
- ⑤ precharge all bitlines.

⇒ Destructive read

Speed

SRAM, 512 Byte → sub-nano sec < 10$/MB
SRAM, KiloByte → a few sec
DRAM, GigaByte → 50 nsec < 1$/MB
Hard, Tera ~ 1 msec < 1$/GB

Hierachy



Memory Locality in temporal, spatial way
- exploit temporal locality
- **Cache** ⇒ (temporal locality principle) ↳ store recently accessed data
  - (Spatial locality) ↳ store data nearby accessed data

Cache hierachy: Lv1 cache, Lv2 cache

Cache - Block, HIT vs Miss, Placement, Replacement Granularity of management, Write Policy, Instruction/data bundle. tag store, Data Store
- Direct-Mapped Cache & Associativity
- LRU
- Shared Cache